

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A219 440

(2)

REPORT DOCUMENTATION PAGE

FORMS
BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
DTIC FILE COPY		
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: Bull HN Information Systems, Inc. GCOS 8 ADA Compilation System, Ver 2.3, DPS 9000 (Host) to DPS 9000 (Target), 890831Sl.10		5. TYPE OF REPORT & PERIOD COVERED 31 Aug. 1989 to 01 Dec. 1990
7. AUTHOR(s) National Institute of Standards and Technology Gaithersburg, Maryland, USA		6. PERFORMING ORG. REPORT NUMBER
8. PERFORMING ORGANIZATION AND ADDRESS National Institute of Standards and Technology Gaithersburg, Maryland, USA		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) National Institute of Standards and Technology Gaithersburg, Maryland, USA		12. REPORT DATE
		13. NUMBER OF PAGES
		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 if different from Report) UNCLASSIFIED		
18. SUPPLEMENTARY NOTES DTIC ELECTE MAR 15 1990 S D & D		
19. KEYWORDS (Continue on reverse side if necessary and identify by block number) Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Bill HN Information Systems, Inc., Gaithersburg, Maryland MD, GCOS 8 ADA Compilation System, Ver 2.3, DPS 9000 under GCOS 8 SR 4000 (Host & Target), ACVC 1.10.		

DD FORM

1473

EDITION OF 1 NOV 65 IS OBSOLETE

1 JAN 73

S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AVF Control Number: NIST89HFS550_1_1.10
22 January 1990

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 890831S1.10146
Bull HN Information Systems, Inc
GCOS 8 ADA Compilation System, Ver 2.3
DPS 9000 Host and DPS 9000 Target

Completion of On-Site Testing:
31 August 1989

Prepared By:
Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

00 03 14 0322

Ada Compiler Validation Summary Report:

Compiler Name: GCOS 8 ADA Compilation System, Ver 2.3

Certificate Number: 890831S1.10146

Host: DPS 9000 under GCOS 8 SR 4000

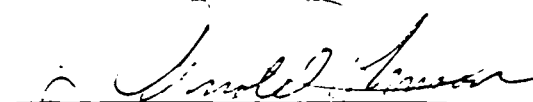
Target: DPS 9000 under GCOS 8 SR 4000

Testing Completed 31 August 1989 Using ACVC 1.10

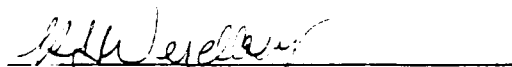
This report has been reviewed and is approved.



Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899



Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899



Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311



Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution	
Availability	
Dist	Avail and/or Special
A-1	



TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT	1-2
1.2	USE OF THIS VALIDATION SUMMARY REPORT	1-2
1.3	REFERENCES	1-3
1.4	DEFINITION OF TERMS	1-3
1.5	ACVC TEST CLASSES	1-4
CHAPTER 2	CONFIGURATION INFORMATION	
2.1	CONFIGURATION TESTED	2-1
2.2	IMPLEMENTATION CHARACTERISTICS	2-1
CHAPTER 3	TEST INFORMATION	
3.1	TEST RESULTS	3-1
3.2	SUMMARY OF TEST RESULTS BY CLASS	3-1
3.3	SUMMARY OF TEST RESULTS BY CHAPTER	3-2
3.4	WITHDRAWN TESTS	3-2
3.5	INAPPLICABLE TESTS	3-2
3.6	TEST, PROCESSING, AND EVALUATION MODIFICATIONS	3-6
3.7	ADDITIONAL TESTING INFORMATION	3-7
3.7.1	Prevalidation	3-7
3.7.2	Test Method	3-7
3.7.3	Test Site	3-8
APPENDIX A	CONFORMANCE STATEMENT	
APPENDIX B	APPENDIX F OF THE Ada STANDARD	
APPENDIX C	TEST PARAMETERS	
APPENDIX D	WITHDRAWN TESTS	
APPENDIX E	COMPILER OPTIONS AS SUPPLIED BY Bull HN Information Systems	

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report. The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- . To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by Gemma Corp under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 31 August 1989 at Bull HN Information Systems.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street,
Washington DC 20301-3081

or from:

Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada	An Ada Commentary contains all information relevant to the Commentary point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the <u>Ada Compiler Validation Procedures and Guidelines</u> .
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure

consistent practices.

Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer which executes the code generated by the compiler.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn	An ACVC test found to be incorrect and not used to check test conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved

words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada

Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated.

A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: GCOS 8 ADA Compilation System, Ver 2.3

ACVC Version: 1.10

Certificate Number: 890831S1.10146

Host Computer:

Machine: DPS 9000

Operating System: GCOS 8 SR 4000

Memory Size: 16 MB

Target Computer:

Machine: DPS 9000

Operating System: GCOS 8 SR 4000

Memory Size: 16 MB

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

Capacities.

- (1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)
- (2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)
- (3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)
- (4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

Predefined types.

- (1) This implementation supports the additional predefined types LONG_INTEGER and LONG_FLOAT in the package STANDARD. (See tests B86001T..Z (7 tests).)

Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

- (1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)
- (2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)
- (3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)
- (4) NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)
- (5) No exception is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

- (6) Underflow is not gradual. (See tests C45524A..Z (26 tests).)

Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

- (1) The method used for rounding to integer is round away from zero. (See tests C46012A..Z (26 tests).)
- (2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..Z (26 tests).)
- (3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

Array types.

An implementation is allowed to raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` for an array having a `'LENGTH` that exceeds `STANDARD.INTEGER'LAST` and/or `SYSTEM.MAX_INT`. For this implementation:

- (1) Declaration of an array type or subtype declaration with more than `SYSTEM.MAX_INT` components raises `NUMERIC_ERROR`. (See test C36003A.)
- (2) `NUMERIC_ERROR` is raised when `'LENGTH` is applied to an array type with `INTEGER'LAST + 2` components. (See test C36202A.)
- (3) `NUMERIC_ERROR` is raised when `'LENGTH` is applied to an array type with `SYSTEM.MAX_INT + 2` components. (See test C36202B.)
- (4) A packed `BOOLEAN` array having a `'LENGTH` exceeding `INTEGER'LAST` raises `NUMERIC_ERROR` when the array type is declared. (See test C52103X.)
- (5) A packed two-dimensional `BOOLEAN` array with more than `INTEGER'LAST` components raises `NUMERIC_ERROR` when the array objects are declared. (See test C52104Y.)
- (6) A null array with one dimension of length greater than `INTEGER'LAST` may raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` either when declared or assigned. Alternatively, an

implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises `NUMERIC_ERROR` when the array type is declared. (See test E52103Y.)

- (7) In assigning one-dimensional array types, the expression is evaluated in its entirety before `CONSTRAINT_ERROR` is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)
- (8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before `CONSTRAINT_ERROR` is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

Discriminated types.

- (1) During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)
- (2) In assigning record types with discriminants, the expression is evaluated in its entirety before `CONSTRAINT_ERROR` is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

Aggregates.

- (1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)
- (2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)
- (3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

Pragmas.

- (1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), ..D (2 tests), and CA3004E..F (2 tests).)

Generics.

- (1) Generic specifications and bodies cannot be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)
- (2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)
- (3) Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)
- (4) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)
- (5) Generic non-library subprogram bodies cannot be compiled in separate compilations from their stubs. (See test CA2009F.)
- (6) Generic package declarations and bodies cannot be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)
- (7) Generic library package specifications and bodies cannot be compiled in separate compilations. (See tests BC3204C and BC3205D.)

- (8) Generic non-library package bodies as subunits cannot be compiled in separate compilations. (See test CA2009C.)

Input and output.

- (1) The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)
- (2) The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)
- (4) Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D..E, CE2102N, and CE2102P.)
- (5) Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I..J (2 tests), CE2102R, CE2102T, and CE2102V.)
- (6) Modes IN_FILE and OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K (3 tests).)
- (7) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)
- (8) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)
- (9) RESET and DELETE operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3112A, and CE3114A.)
- (10) Overwriting to a sequential file truncates to the last element written. (See test CE2208B.)
- (11) Temporary sequential files are not given names and not deleted when closed. (See test CE2108A.)
- (12) Temporary direct files are not given names and not deleted when closed. (See test CE2108C.)
- (13) Temporary text files are not given names and not deleted when closed. (See test CE3112A.)
- (14) Only one internal file can be associated with each external file for sequential files when writing or reading. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)

- (15) Only one internal file can be associated with each external file for direct files when writing or reading. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)
- (16) Only one internal file can be associated with each external file for text files when writing or reading (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 520 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 173 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 61 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	121	1129	1816	17	24	46	3153
Inapplicable	8	9	499	0	4	0	520
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER														TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14		
Passed	200	581	567	245	172	99	159	331	135	36	250	99	279	3153	
Inapplicable	12	68	113	3	0	0	7	1	2	0	2	270	42	520	
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44	
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717	

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

A39005G	B97102E	C97116A	BC3009B	CD2A62D	CD2A63A
CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B	CD2A66C
CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D	CD2A76A
CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G	CD2A84M
CD2A84N	CD2B15B	CD2B15C	CD2D11B	CD5007B	CD50110
CD7105A	CD7203B	CD7204B	CD7205C	CD7205D	CE2107I
CE3111C	CE3301A	CE3411B	E28005C	ED7004B	ED7005C
ED7006C	ED7006D				

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 520 tests were inapplicable for the reasons indicated:

The following 173 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113N..Y (12 tests)	C35705N..Y (12 tests)
C35706N..Y (12 tests)	C35707N..Y (12 tests)
C35708N..Y (12 tests)	C35802N..Z (13 tests)
C45241N..Y (12 tests)	C45321N..Y (12 tests)
C45421N..Y (12 tests)	C45521N..Z (13 tests)
C45524N..Z (13 tests)	C45621N..Z (13 tests)
C45641N..Y (12 tests)	C46012N..Z (13 tests)

The following 170 tests are not applicable because 'SIZE representation clauses are not supported:

A39005B	C87B62A	CD1009A..I (9 tests)
CD1009O..Q (3 tests)	CD1C03A	CD1C04A
CD2A21A..E (5 tests)	CD2A22A..J (10 tests)	
CD2A23A..E (5 tests)	CD2A24A..J (10 tests)	
CD2A31A..D (4 tests)	CD2A32A..J (10 tests)	
CD2A41A..E (5 tests)	CD2A42A..J (10 tests)	
CD2A51A..E (5 tests)	CD2A52A..D (4 tests)	
CD2A52G..J (4 tests)	CD2A53A..E (5 tests)	
CD2A54A..D (4 tests)	CD2A54G..J (4 tests)	
CD2A61A..L (12 tests)	CD2A62A..C (3 tests)	
CD2A64A..D (4 tests)	CD2A65A..D (4 tests)	
CD2A71A..D (4 tests)	CD2A72A..D (4 tests)	
CD2A74A..D (4 tests)	CD2A75A..D (4 tests)	
CD2A81A..E (5 tests)	CD2A81F	
CD2A83A..C (3 tests)	CD2A83E..F (2 tests)	
CD2A84B..I (8 tests)	CD2A84K..L (2 tests)	
CD2A87A	CD2A91A..E (5 tests)	
ED2A26A	ED2A56A	ED2A86A

The following 7 tests are not supported because 'SMALL representation clauses are not supported:

A39005E	C87B62C	CD1009L	CD1C03F	CD1C04C	CD2D11A
CD2D13A					

C35508I, C35508J, C35508M, and C35508N are not applicable because they include enumeration representation clauses for BOOLEAN types in which the representation values are other than (FALSE => 0, TRUE => 1). Under the terms of AI-00325, this implementation is not required to support such representation clauses.

C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

The following 16 tests are not applicable because this implementation does not support a predefined type `SHORT_INTEGER`:

C45231B	C45304B	C45502B	C45503B	C45504B
C45504E	C45611B	C45613B	C45614B	C45631B
C45632B	B52004E	C55B07B	B55B09D	B86001V
CD7101E				

B86001X, C45231D, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than `INTEGER`, `LONG_INTEGER`, or `SHORT_INTEGER`.

B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than `FLOAT`, `LONG_FLOAT`, or `SHORT_FLOAT`.

B86001Y is not applicable because this implementation supports no predefined fixed-point type other than `DURATION`.

C4A013B is not applicable because the evaluation of an expression involving `'MACHINE_RADIX` applied to the most precise floating-point type would raise an exception; since the expression must be static, it is rejected at compile time.

The following 76 tests are not applicable because, for this implementation, type `SYSTEM.ADDRESS` is a limited private type:

CD5003B..I (8 tests)	CD5011A..I (9 tests)
CD5011K..M (4 tests)	CD5011Q..S (3 tests)
CD5012A..J (10 tests)	CD5012L..M (2 tests)
CD5013A..I (9 tests)	CD5013K..O (5 tests)
CD5013R..S (2 tests)	CD5014A..O (15 tests)
CD5014R..Z (9 tests)	

C96005B is not applicable because there are no values of type `DURATION'BASE` that are outside the range of `DURATION`.

CA2009C is not applicable because this implementation does not permit compilation of generic non-library package bodies as subunits in separate files from their stubs.

CA2009F is not applicable because this implementation does not permit compilation of generic non-library subprogram bodies as subunits in separate files from their stubs.

BC3204C and BC3205D are not applicable because this implementation does not permit compilation of generic library package bodies in different files from their specifications.

The following 19 tests are not applicable because no representation clauses may be given for a derived type:

AD1C04D	AD3015C	AD3015F	AD3015H	AD3015K
CD1C04B	CD1C04E	CD3015A	CD3015B	CD3015D
CD3015E	CD3015G	CD3015I	CD3015J	CD3015L
CD4051A..D (4 tests)				

AE2101H and EE2401D use instantiations of package `DIRECT_IO` with unconstrained array types. These instantiations are rejected by this compiler.

CE2102E is inapplicable because this implementation supports `CREATE` with `OUT_FILE` mode for `SEQUENTIAL_IO`.

CE2102F is inapplicable because this implementation supports `CREATE` with `INOUT_FILE` mode for `DIRECT_IO`.

CE2102I is inapplicable because this implementation supports `CREATE` with `IN_FILE` mode for `DIRECT_IO`.

CE2102J is inapplicable because this implementation supports `CREATE` with `OUT_FILE` mode for `DIRECT_IO`.

CE2102N is inapplicable because this implementation supports `OPEN` with `IN_FILE` mode for `SEQUENTIAL_IO`.

CE2102O is inapplicable because this implementation supports `RESET` with `IN_FILE` mode for `SEQUENTIAL_IO`.

CE2102P is inapplicable because this implementation supports `OPEN` with `OUT_FILE` mode for `SEQUENTIAL_IO`.

CE2102Q is inapplicable because this implementation supports `RESET` with `OUT_FILE` mode for `SEQUENTIAL_IO`.

CE2102R is inapplicable because this implementation supports `OPEN` with `INOUT_FILE` mode for `DIRECT_IO`.

CE2102S is inapplicable because this implementation supports `RESET` with `INOUT_FILE` mode for `DIRECT_IO`.

CE2102T is inapplicable because this implementation supports `OPEN` with `IN_FILE` mode for `DIRECT_IO`.

CE2102U is inapplicable because this implementation supports `RESET` with `IN_FILE` mode for `DIRECT_IO`.

CE2102V is inapplicable because this implementation supports `OPEN` with `OUT_FILE` mode for `DIRECT_IO`.

CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

CE2105A is inapplicable because CREATE with IN_FILE mode is not supported by this implementation for SEQUENTIAL_IO.

CE2107A..H (8 tests), CE2107L, CE2110B, CE2110D, CE2111D and CE2111H are not applicable because multiple internal files cannot be associated with the same external file for sequential files or direct files. The proper exception is raised when multiple access is attempted.

CE3102F is inapplicable because text file RESET is supported by this implementation.

CE3102G is inapplicable because text file DELETE is supported by this implementation.

CE3102I is inapplicable because text file CREATE with OUT_FILE mode is supported by this implementation.

CE3102J is inapplicable because text file OPEN with IN_FILE mode is supported by this implementation.

CE3102K is inapplicable because text file OPEN with OUT_FILE mode is not supported by this implementation.

CE3109A is inapplicable because text file CREATE with IN_FILE mode is not supported by this implementation.

CE3111A..B (2 tests), CE3111D..E (2 tests), CE3114B, and CE3115A are not applicable because multiple internal files cannot be associated with the same external file for text files. The proper exception is raised when multiple access is attempted.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 61 tests.

The following tests were split because syntax errors at one point

resulted in the compiler not detecting other errors in the test:

B22003A	B26001A	B26002A	B26005A	B28001D	B28003A
B29001A	B33001B	B35101A	B37106A	B37301B	B37302A
B38003A	B38003B	B38009A	B38009B	B51001A	B53009A
B54A01C	B54A01J	B55A01A	B61001C	B61001D	B61001F
B61001H	B61001I	B61001M	B61001R	B61001W	B67001H
B91001A	B91002A	B91002B	B91002C	B91002D	B91002E
B91002F	B91002G	B91002H	B91002I	B91002J	B91002K
B91002L	B95030A	B95061A	B95061F	B95061G	B95077A
B97103E	B97104G	BA1101B	BC1109A	BC1109C	BC1109D
BC1202A	BC1202B	BC1202E	BC1202F	BC1202G	BC2001D
BC2001E					

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the GCOS 8 ADA Compilation System compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the GCOS 8 ADA Compilation System compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	DPS 9000
Host operating system:	GCOS 8 SR 4000
Target computer:	DPS 9000
Target operating system:	GCOS 8 SR 4000

A tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precision was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized on site. Tests requiring modifications during the prevalidation testing were included in their modified form on the tape.

TEST INFORMATION

The contents of the tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the DPS 9000. Results were printed from the host computer.

The compiler was tested using command scripts provided by Bull HN Information Systems Inc. and reviewed by the validation team. The compiler was tested using all default option settings. See Appendix E for a complete listing of the compiler options for this implementation.

Tests were compiled, linked, and executed (as appropriate) using one host computer and one target computer. Test output, compilation listings, and job logs were captured on tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Bull HN Information Systems Inc. and was completed on 31 August 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

Bull HN Information Systems Inc. has submitted the following Declaration of Conformance concerning the GCOS 8 ADA Compilation System.

Declaration of Conformance

Compiler Implementor: Bull HN Information Systems Inc.
Ada Validation Facility: National Institute of Standards and Technology
Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configurations

Base Compiler Name:	GCOS 8 Ada Compilation System	Version:	2.3
Host Architecture:	DPS 8000	OS&VER #:	GCOS 8 SR 3000
Target Architecture:	DPS 8000	OS&VER #:	GCOS 8 SR 3000
Host Architecture:	DPS 9000	OS&VER #:	GCOS 8 SR 4000
Target Architecture:	DPS 9000	OS&VER #:	GCOS 8 SR 4000

Derived Compiler Registrations

Derived Compiler Name:	GCOS 8 Ada Compilation System	Version:	2.3
Host Architecture:	DPS 8/70	OS&VER #:	GCOS 8 SR 3000
Target Architecture:	DPS 8/70	OS&VER #:	GCOS 8 SR 3000
Host Architecture:	DPS 8000	OS&VER #:	GCOS 8 SR 4000
Target Architecture:	DPS 8000	OS&VER #:	GCOS 8 SR 4000
Host Architecture:	DPS 90	OS&VER #:	GCOS 8 SR 4000
Target Architecture:	DPS 90	OS&VER #:	GCOS 8 SR 4000

Implementer's Declaration

I, the undersigned, representing Bull HN Information Systems Inc., have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that Bull HN Information Systems Inc. is the owner of record of the Ada language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI-MIL-STD-1815A. All certificates and registrations for the Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

David Mosley
Bull HN Information Systems Inc.
David Mosley, Consulting Software Engineer

Date: 8/21/89

Owner's Declaration

I, the undersigned, representing Bull HN Information Systems Inc., take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that the Ada language compiler listed, and its host/target performance is in compliance with the Ada Language Standard ANSI/MIL-STD-1815A. I have reviewed the Validation Summary Report for the compiler and concur with the contents.

David Curry
Bull HN Information Systems Inc.
David Curry, Manager
Production Environment Development Products

Date: 8/22/89

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the GCOS 8 ADA Compilation System, as described in this Appendix, are provided by Bull HN Information Systems. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

package STANDARD is

...

```
type INTEGER is range -34359738368 .. 34359738367;
type LONG_INTEGER is range -23611832241434822606848 ..
                        23611832241434822606847;
```

```
type FLOAT is digits 6 range -16#0.1E128 .. 16#0.FFFFFFFE#E127;
type LONG_FLOAT is digits 17 range -16#0.1#E128 ..
                        16#0.FFFFFFFFFFFFFFFFFE#E127;
```

```
type DURATION is delta 0.000016 range -1099799511627776.0 ..
                        1099799511627776.0;
```

...

end STANDARD;

5 Appendix F of the Ada Language Reference Manual

1. Implementation-dependent Pragmas

Pragma INTERFACE SPELLING

This pragma has the form:

```
pragma INTERFACE_SPELLING (subprogram_name, string_literal);
```

The name of a subprogram written in another language may be an illegal identifier within the Ada language. This pragma can be used to establish a correspondence between the external name (given by the `string_literal`) and the Ada identifier (`subprogram_name`) specified in a `pragma INTERFACE`.

This pragma is allowed at the place of a declarative item and must apply to a subprogram declared by an earlier `pragma INTERFACE`.

Pragma MULTI SEGMENT DATA

This pragma has the form:

```
pragma MULTI_SEGMENT_DATA;
```

This pragma affects the method used for static data storage access and, hence, the amount of static storage which is accessible to this compilation unit. By specifying a more costly access method, this pragma allows the compilation unit to access over 256K words of static storage (the default limit on static storage is 32K words).

This pragma is allowed anywhere a pragma is allowed. It will affect the current compilation unit and any subsequent units in the same compilation.

Pragma SINGLE SEGMENT DATA

This pragma has the form:

```
pragma SINGLE_SEGMENT_DATA;
```

This pragma affects the method used for static data storage access and, hence, the amount of static storage which is accessible to this compilation unit. By specifying a more costly access method, this pragma allows the compilation unit to access up to 256K words of static storage (the default limit on static storage is 32K words).

This pragma is allowed anywhere a pragma is allowed. It will affect the current compilation unit and any subsequent units in the same compilation.

2. Implementation-dependent Attributes

There are no implementation-dependent attributes.

3. Package System

The specification of the package SYSTEM is as follows:

package SYSTEM is

type ADDRESS is access INTEGER;

subtype PRIORITY is INTEGER range 1..15;

type NAME is (DPS8);

SYSTEM_NAME : constant NAME := DPS8;

STORAGE_UNIT : constant := 36;

MEMORY_SIZE : constant := 256*1024;

MIN_INT : constant := -2361_183241_434822_606848;

MAX_INT : constant := 2361_183241_434822_606847;

MAX_DIGITS : constant := 17;

MAX_MANTISSA : constant := 60;

FINE_DELTA : constant := 2.0**(-60);

TICK : constant := 0.000016;

type INTERFACE_LANGUAGE is (GMAPV, COBOL_85, PL_6, FORTRAN_77, C,
GMAPV_ADA);

end SYSTEM;

4. Representation Clauses

The list of all restrictions on representation clauses is as follows:

In general, no type representation clauses may be given for a derived type. The type representation clauses that are accepted for non-derived types are described in the following:

The compiler accepts only a length clause that specifies the number of storage units to be reserved for a collection and the number of storage units reserved for an activation of a task.

Enumeration representation clauses may specify representations only in the range of the predefined type INTEGER.

Record representation clauses do not support alignment clauses. A component clause is allowed if, and only if, either of these statements is true:

- The component type is a discrete type different from LONG_INTEGER.
- The component type is an array type with a discrete element type different from LONG_INTEGER.

No component clause is allowed if the component type is not covered by the above two inclusions. If the record type contains components not covered by a component clause, they are allocated consecutively after the component with

the highest AT value. Allocation of a record component without a component clause is always aligned on a storage unit boundary. Holes created because of component clauses are not otherwise utilized by the compiler.

5. Implementation-generated Names for Implementation-dependent Components

There are no implementation-generated names for implementation-dependent components.

6. Address Clauses

Address clauses are not supported.

7. Unchecked Conversion

Unchecked conversion is allowed only between values of the same size. For dynamic arrays this will be checked at runtime. Unchecked conversion between types where at least one is an unconstrained array type is not allowed.

8. Input-Output

The implementation-dependent characteristics of the input-output packages are as follows:

An attempt to CREATE an IN_FILE will raise USE_ERROR.

A RESET to OUT_FILE, on a sequential or text file, empties the file.

Two internal files may not be associated with the same external file simultaneously.

Temporary files accessed by a batch program are not named. Use of the function NAME results in a USE_ERROR.

An attempt to open a "busy" file will result in the I/O exception, STATUS_ERROR.

The FORM parameter can be the concatenation of any of the following strings separated by spaces:

- "-FILCOD XX" Allows the association of an internal file to a file code where XX is a valid file code supplied by JCL. A file code takes precedence over a NAME string.
- "-MEDCOD N" Allows the specification of the media code of the external file where N is a valid media code.
- "-APPEND" On opening a file with mode OUT_FILE, allows the positioning at end-of-file so that writes will append rather than overwrite the file. It has no effect on creates or on opening files with mode IN_FILE. It is only applicable to sequential

and text files; it results in a USE_ERROR if it is applied to direct files.

A file name may consist of 10 levels of qualifications: up to 9 catalog names and a file name. Each name must be no more than 12 characters in length. Passwords are not currently implemented. If only a file name is given in the name parameter, it is prefaced by the userid of the process in execution.

All Ada files are mapped to GCOS system standard files. Blocks are of variable length with a maximum size of 320 words. Each block has a control word containing the block size and a sequence number. Records are of variable format with record control words, but in a direct or sequential file, all record elements are the same size. Records may span as many physical blocks as is necessary.

Ada files are recorded in ASCII format and are compatible with standard time sharing tools. When a BCD file is read (media codes 0, 2, 3), each record is translated to ASCII format as it is passed to the user's record area. There is no physical difference between the formats of direct and sequential files and a sequentially created file may be read directly and vice versa.

The use of protected files should be avoided. A program cannot define commitment points: therefore it can adversely impact the performance of other processes accessing the protected files.

The text input/output provides input and output of data in human-readable form. The physical file format of disk files is the same as for sequential I/O except that the records may vary in length from null to 4096 bytes.

The following applies only to files on disk. The standard input and output files are described later. Line terminators and file terminators are not explicitly present on the file. A line is a variable length record and can contain up to 4096 bytes including a possible page terminator. A page terminator is a form feed character (FF, ASCII 12) and is always appended to the last line of a page.

If more than 4096 characters are input, a USE_ERROR is raised. The last record of a file will not contain a form feed character since the physical end-of-file will signal both page and file termination.

The standard input and output files are equated to file codes I* and P*, respectively. They may be redirected to the terminal by setting bit 19 in the program switch word (\$ SET 19). The program can then be connected via its SNLMB using the time sharing JDAC commands.

The file code P* will be written as media code 7 (ASCII print) and it may be assigned to SYSOUT or to a disk file. Any horizontal tab character is converted to a space. A single space control sequence is appended to the end of every line. A page eject sequence is placed at the end of every page. These control characters must be accounted for if a disk file is to be reread.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

8 Macro Definitions

```
ACC_SIZE                : 36
BIG_ID1                 : (1..248 => 'A', '1')
BIG_ID2                 : (1..248 => 'A', '2')
BIG_ID3                 : (1..124 => 'A', '3', 126..249 => 'A')
BIG_ID4                 : (1..124 => 'A', '4', 126..249 => 'A')
BIG_INT_LIT             : (1..246 => 0, 298)
BIG_REAL_LIT            : (1..244 => 0, 690.0)
BIG_STRING1             : (1..124 => 'A')
BIG_STRING2             : (125..248 => 'A', '1')
BLANKS                  : (1..229 => ' ')
COUNT_LAST             : 34359738367
DEFAULT_MEM_SIZE        : 262144
DEFAULT_STOR_UNIT       : 36
DEFAULT_SYS_NAME        : DPS8
DELTA_DOC               : 2#1.0#E-60
FIELD_LAST              : 75
FIXED_NAME              : NO_SUCH_FIXED_TYPE
FLOAT_NAME              : NO_SUCH_FLOAT_TYPE
GREATER_THAN_DURATION   : 0.0
GREATER_THAN_DURATION_BASE_LAST : 1100000000000.0
HIGH_PRIORITY           : 15
ILLEGAL_EXTERNAL_FILE_NAME1 : F@LENAME
ILLEGAL_EXTERNAL_FILE_NAME2 : NAMETOLONGFORFILENAME
INTEGER_FIRST           : -34359738368
INTEGER_LAST            : 34359738367
INTEGER_LAST_PLUS_1     : 34359738368
LESS_THAN_DURATION      : 0.0
LESS_THAN_DURATION_BASE_FIRST : -1100000000000.0
LOW_PRIORITY            : 1
MANTISSA_DOC            : 60
MAX_DIGITS              : 17
MAX_INT                 : 2361183241434822606847
MAX_INT_PLUS_1          : 2361183241434822606848
MAX_IN_LEN              : 249
MAX_LEN_INT_BASED_LITERAL : (2:, 3..246 => '0', 11:)
MAX_LEN_REAL_BASED_LITERAL : (16:, 4..245 => '0', F.E:)
MAX_STRING_LITERAL      : ('', 2..248 => 'A', '')
MIN_INT                 : -2361183241434822606848
MIN_TASK_SIZE           : 36
NAME                    : NO_SUCH_INTEGER_TYPE
NAME_LIST               : DPS8
NEG_BASED_INT           : 16#FFFFFFFFFFFFFFFFFFFFF#
NEW_MEM_SIZE            : 262144
NEW_STOR_UNIT           : 36
NEW_SYS_NAME            : DPS8
TASK_SIZE               : 36
TICK                    : 0.000016
```

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

A39005G

This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E

This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

C97116A

This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

BC3009B

This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D

This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]

These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84N & M, & CD50110

These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C

These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B

This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B

This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]

These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A

This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK -- particular instances of change may be less (line 29).

CD7203B, & CD7204B

These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D

This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I

This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C

This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A

This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B

This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

E28005C

This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

APPENDIX E

COMPILER OPTIONS AS SUPPLIED BY

Bull HN Information Systems Inc.

Compiler: GCOS 8 ADA Compilation System, Ver 2.3

ACVC Version: 1.10

9 Testing Method

- A) The validation tape will be loaded directly onto the host machine (DPS 90). After necessary modifications are made, GCOS save tapes will be made of the ACVC sources. These save tapes will be loaded onto the other host machines: DPS 8/70, DPS 8000, and DPS 9000.
- B) The command files can be found on the enclosed tape, see section 11.
- C) All tests were run with the same compiler options: namely:

BOTTOM=3 (default)	Specify no. of lines in listing bottom margin
ERROR_LIMIT=200 (default)	Specify max. no. of errors in compilation
INPUT=249	Specify max. no. of chars in source line
LIBRARY= <i>library_name</i>	Specify current sublibrary
LIST	Create listing
NO_DEBUG (default)	Don't generate debugging data
NO_LIST_OUT (default)	Don't produce object code listing
NO_XREF (default)	Don't produce cross-reference listing
OUTPUT=100 (default)	Specify max. no. of chars in listing line
PAGE=55 (default)	Specify max. no. lines in listing page
PROGRESS_REPORT	Print passes of compiler on execution report
SOURCE_FILE= <i>file_name</i>	Specify source file
TOP=5 (default)	Specify no. of lines in listing top margin

The following additional compiler options were available:

DEBUG	Generate debugging data
DUMP	Produce complete dump on abort
LIST_OUT	Produce object code listing
MSDATA	Allow for more than 256K words of static data
NO_ACCESS_CHECK	Suppress runtime checks on access values
NO_CHECKS	Suppress all runtime checks
NO_DISCRIMINANT_CHECK	Suppress runtime checks on discriminant values
NO_DIVISION_CHECK	Suppress runtime checks on division by zero
NO_ELABORATION_CHECK	Suppress runtime checks on elaboration
NO_INDEX_CHECK	Suppress runtime checks on index values
NO_LENGTH_CHECK	Suppress runtime checks on matching components
NO_LIST	Don't produce listing
NO_OVERFLOW_CHECK	Suppress runtime checks on overflow
NO_RANGE_CHECK	Suppress runtime checks on range constraints
NO_STORAGE_CHECK	Suppress runtime checks on storage allocation
OPTIMIZE_ALL	Enable all optimizations
OPTIMIZE_BLOCK	Enable block optimization
OPTIMIZE_CHECK	Enable check optimization
OPTIMIZE_CSE	Enable common subexpression optimization
OPTIMIZE_FCT2PROC	Enable function optimization
OPTIMIZE_PEEP	Enable peephole optimization
OPTIMIZE_REORDER	Enable expression reordering optimization
OPTIMIZE_STACK_HEIGHT	Enable stack height optimization
SAVE_SOURCE	Save source copy in current sublibrary
SCC= <i>string</i>	Specify security classification code
SSDATA	Allow for more than 32K words of static data
STANDARD	Compile package STANDARD
TEST= <i>string</i>	Set compiler test switches
UNIT_ID= <i>integer</i>	Specify unit number of compilation unit
XREF	Produce cross-reference listing

D) The host and target are not different.